CS 577: NATURAL LANGUAGE PROCESSING

Abulhair Saparov

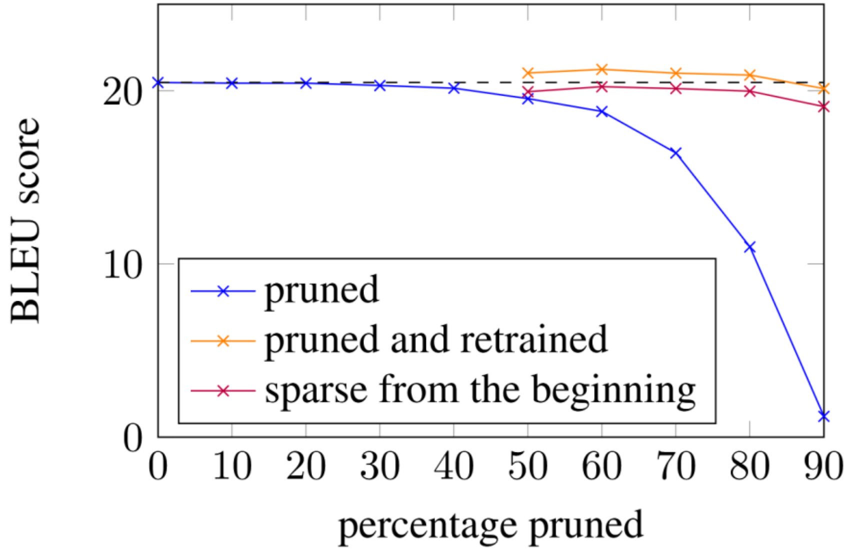
Lecture 16: Pruning

MODEL COMPRESSION

- Last time, we discussed quantization as a method for model compression.
- There are two additional high-level approaches to model compression:
  - Pruning: Remove parts of the model while minimizing any adverse effects on model performance.
  - Distillation: Use a larger model to train a small model.
- We will focus on pruning today.

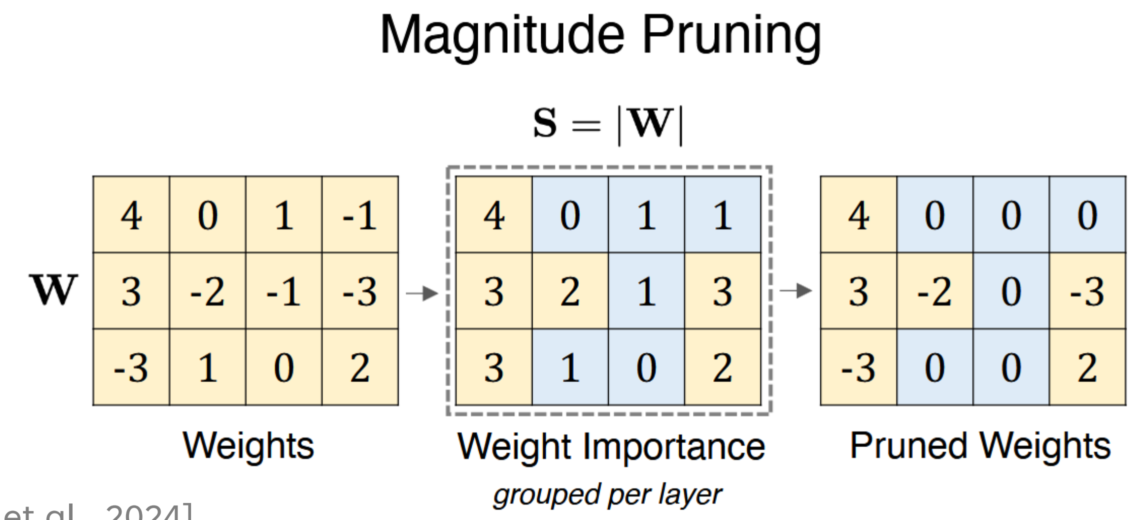
MAGNITUDE PRUNING

- Perhaps the simplest pruning approach for neural models is magnitude pruning (Han et al., 2015, See et al., 2016).
- Select $X\%$ of the model parameters with the smallest magnitude and set them to zero.
- See et al. (2016) trained a NMT model (English-German) on WMT’14 dataset:



MAGNITUDE PRUNING

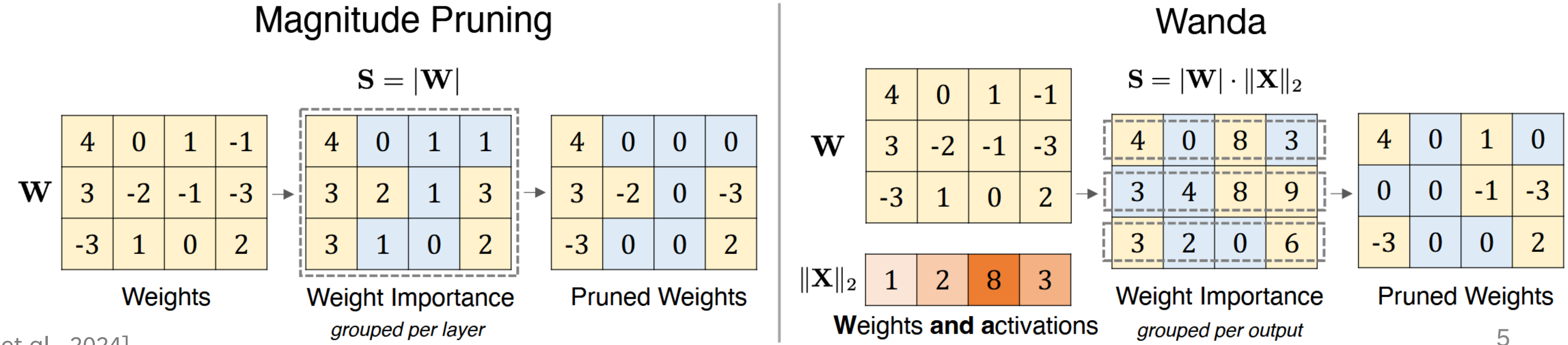
- One disadvantage of magnitude pruning is that it ignores activations.
- For example, if a small weight value is frequently multiplied by large activations, then the weight may be important.
- Similarly, a large weight may be frequently multiplied by very small activations.



[Sun et al., 2024]

WANDA

- An alternative approach is to use a set of calibration data X to compute the activations in a forward pass of the model.
- We multiply the magnitude of the activations with the weights to obtain a "weight importance" metric.
- We prune the weights with the smallest importance.



[Sun et al., 2024]

WANDA

- This approach is called Wanda (Pruning by **W**eights **and A**ctivations; Sun et al., 2024).
- They evaluated their model on 7 zero-shot tasks and computed the average accuracies:

			LLaMA				LLaMA-2		
Method	Weight Update	Sparsity	7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	59.99	62.59	65.38	66.97	59.71	63.03	67.08
Magnitude	✗	50%	46.94	47.61	53.83	62.74	51.14	52.85	60.93
SparseGPT	✓	50%	**54.94**	58.61	63.09	66.30	**56.24**	60.72	**67.28**
Wanda	✗	50%	54.21	**59.33**	**63.60**	**66.67**	**56.24**	**60.83**	67.03

[Sun et al., 2024]

WANDA

- This approach is called Wanda (Pruning by **W**eights **and A**ctivations; Sun et al., 2024).
- They also evaluated the language modeling perplexity on the `WikiText` dataset.

			LLaMA				LLaMA-2		
Method	Weight Update	Sparsity	7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	5.68	5.09	4.77	3.56	5.12	4.57	3.12
Magnitude	✗	50%	17.29	20.21	7.54	5.90	14.89	6.37	4.98
SparseGPT	✓	50%	**7.22**	6.21	5.31	**4.57**	6.51	5.63	**3.98**
Wanda	✗	50%	7.26	**6.15**	**5.24**	**4.57**	**6.42**	**5.56**	**3.98**

[Sun et al., 2024]

WHY IS PRUNING HELPFUL?

- So what if we zero out some values in parameter matrices?
  - How does this help if the overall sizes of the matrices is unchanged?
  - Isn't the cost of the matrix multiplication unchanged?
- Yes, if we use dense matrix multiplication algorithms.
- Consider the representation of a vector $v \in \mathbb{R}^n$.
- In a dense representation, we simply store an array of floating-point values.
  - The length of the array does not change even if many of the values are 0.
- In a sparse representation, we can store a sorted list of tuples (i, v_i),
  - where the first value in the tuple stores an index i,
  - and the second value stores the value of the vector at that index v_i.

SPARSE VECTORS, MATRICES, TENSORS

- For example, if we have the dense vector:

```
[0.0, -4.1, 6.2, 0.0, 0.0, -0.7, 0.0, 0.0, 0.0, 1.9, 0.0, 0.0]
```

- We can store a list of the only the non-zero elements:

```
indices: [1, 2, 5, 9]
values:  [-4.1, 6.2, -0.7, 1.9]
```

- This representation can be extended to matrices and tensors as well:

$$\mathbf{A} = \begin{bmatrix} 1.0 & 0 & 0 & 2.0 & 0 \\ 3.0 & 4.0 & 0 & 5.0 & 0 \\ 6.0 & 0 & 7.0 & 8.0 & 9.0 \\ 0 & 0 & 10.0 & 11.0 & 0 \\ 0 & 0 & 0 & 0 & 12.0 \end{bmatrix} \longrightarrow$$

$$\begin{aligned} \text{data} &= [12.0 \quad 9.0 \quad 7.0 \quad 5.0 \quad 1.0 \quad 2.0 \quad 11.0 \quad 3.0 \quad 6.0 \quad 4.0 \quad 8.0 \quad 10.0] \\ \text{row} &= [4 \quad 2 \quad 2 \quad 1 \quad 0 \quad 0 \quad 3 \quad 1 \quad 2 \quad 1 \quad 2 \quad 3] \\ \text{col} &= [4 \quad 4 \quad 2 \quad 3 \quad 0 \quad 3 \quad 3 \quad 0 \quad 0 \quad 1 \quad 3 \quad 2] \end{aligned}$$

[UIUC Course Staff, CS 357 Textbook, 2025]

SPARSE VECTORS, MATRICES, TENSORS

- This matrix/tensor representation is called coordinate format (COO).

$$\mathbf{A} = \begin{bmatrix} 1.0 & 0 & 0 & 2.0 & 0 \\ 3.0 & 4.0 & 0 & 5.0 & 0 \\ 6.0 & 0 & 7.0 & 8.0 & 9.0 \\ 0 & 0 & 10.0 & 11.0 & 0 \\ 0 & 0 & 0 & 0 & 12.0 \end{bmatrix} \longrightarrow \begin{aligned} \text{data} &= [12.0 \quad 9.0 \quad 7.0 \quad 5.0 \quad 1.0 \quad 2.0 \quad 11.0 \quad 3.0 \quad 6.0 \quad 4.0 \quad 8.0 \quad 10.0] \\ \text{row} &= [4 \quad 2 \quad 2 \quad 1 \quad 0 \quad 0 \quad 3 \quad 1 \quad 2 \quad 1 \quad 2 \quad 3] \\ \text{col} &= [4 \quad 4 \quad 2 \quad 3 \quad 0 \quad 3 \quad 3 \quad 0 \quad 0 \quad 1 \quad 3 \quad 2] \end{aligned}$$

[UIUC Course Staff, CS 357 Textbook, 2025]

SPARSE VECTORS, MATRICES, TENSORS

- This matrix/tensor representation is called coordinate format (COO).
- There are other sparse representations such as compressed sparse row (CSR, or Yale format).
    - Here, non-zero elements in the matrix are stored in a vector in row-major sorted order.
    - `rowptr` stores the indices within `data` that mark the beginning of each row.
        - (`rowptr` has length equal to the number of rows)
        - `col` specifies the column of each element of `data`.

$$\mathbf{A} = \begin{bmatrix} 1.0 & 0 & 0 & 2.0 & 0 \\ 3.0 & 4.0 & 0 & 5.0 & 0 \\ 6.0 & 0 & 7.0 & 8.0 & 9.0 \\ 0 & 0 & 10.0 & 11.0 & 0 \\ 0 & 0 & 0 & 0 & 12.0 \end{bmatrix} \longrightarrow$$

$$\text{data} = [1.0 \quad 2.0 \quad 3.0 \quad 4.0 \quad 5.0 \quad 6.0 \quad 7.0 \quad 8.0 \quad 9.0 \quad 10.0 \quad 11.0 \quad 12.0]$$

$$\text{col} = [0 \quad 3 \quad 0 \quad 1 \quad 3 \quad 0 \quad 2 \quad 3 \quad 4 \quad 2 \quad 3 \quad 4]$$

$$\text{rowptr} = [0 \quad 2 \quad 5 \quad 9 \quad 11 \quad 12]$$

[UIUC Course Staff, CS 357 Textbook, 2025]

SPARSE VECTORS, MATRICES, TENSORS

- Consider the vector dot-product operation for dense vs sparse vectors.
- For dense vector dot-product,
  - There will be n multiplication operations and n addition operations,
  - For a total of $2n$ operations.
- For sparse vector dot-product (in COO),
  - We scan through the two ordered lists, only multiplying elements with matching indices.
  - In the worst case, both vectors will have the same non-zero elements.
  - So we need $2k$ comparison operations, k multiplication operations, and k addition operations.
  - For a total of $4k$ operations.

SPARSE VECTORS, MATRICES, TENSORS

- Thus, if the number of nonzero elements is sufficiently small, sparse vector/matrix operations will be faster than their dense counterparts,
  - Even if the vectors/matrices have the same dimensions.
- The ratio of zero elements to total elements ($1 - k/n$) is called the sparsity.
- What level of sparsity do we need for sparse operations to be faster than dense operations?

SPARSE VECTORS, MATRICES, TENSORS

- Consider a simple Python benchmark to measure the speed of sparse vs dense matrix multiplication:

```
from scipy import sparse
import numpy as np
from time import perf_counter

def test(n, sparsity):
        # create two random n x n matrices with given `sparsity`
        A = sparse.random_array(shape=(n,n), density=(1.0 - sparsity), dtype=np.float32)
        B = sparse.random_array(shape=(n,n), density=(1.0 - sparsity), dtype=np.float32)

        # measure the duration of sparse matrix multiplication
        time_start = perf_counter()
        C = A @ B
        sparse_time = perf_counter() - time_start

        # convert the matrices to dense format
        A = A.toarray()
        B = B.toarray()

        # measure the duration of dense matrix multiplication
        time_start = perf_counter()
        C = A @ B
        dense_time = perf_counter() - time_start
        return (sparse_time, dense_time)
```

SPARSE VECTORS, MATRICES, TENSORS

- Consider a simple Python benchmark to measure the speed of sparse vs dense matrix multiplication.
- Run on AMD EPYC 7543 CPU.
- For sparsity = `0.5`, and matrix dimensions 1000×1000, each matrix product takes time: (averaged over 50 trials)

```
sparse time (ms): 436.3604660797864
dense time (ms):  713.994812448509
```

- For sparsity = `0.9`:

```
sparse time (ms): 51.92083204397932
dense time (ms):  727.737107896246
```

SPARSE VECTORS, MATRICES, TENSORS

- But these benchmarks were run on CPU.
- If we instead use PyTorch (package `torch.sparse`) to run the dense matrix product on an A30 GPU,
- For sparsity = `0.9` and 1000 trials:

```
sparse time (ms): 3.1558132271748036
dense time (ms):  0.24804117833264172
```

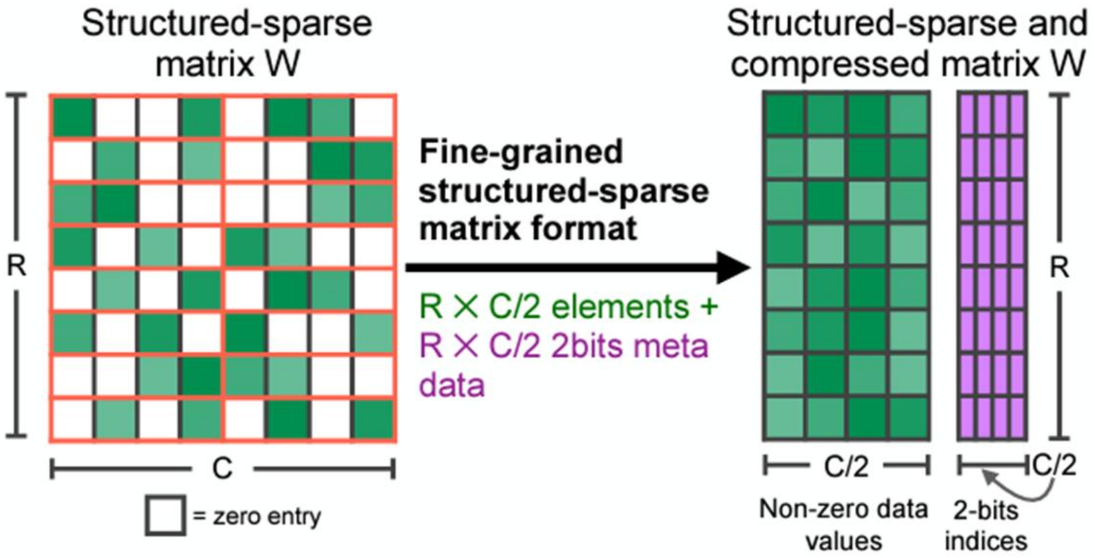
- GPUs are much better optimized to parallelize dense operations.
- But even if this sparse representation does not improve computation speed,
  - It does save memory,
  - With memory savings increasing linearly with sparsity.
- Maybe with better hardware optimizations, this representation may become faster in the future. (active area of research)

UNSTRUCTURED VS STRUCTURED PRUNING

- But can we use a different representation of sparse vectors/matrices that would enable faster computation?
- Magnitude pruning is an example of unstructured pruning,
  - Where there is no constraint on the positions of the pruned weights within each vector/matrix.
- What if we add a requirement that every n contiguous elements in the vector/matrix must contain at least k zeros?
- This representation is called $k{:}n$ sparsity or $k{:}n$ structured sparsity.
  - As suggested in the name, this sparse representation is an example of structured sparsity.
  - Pruning methods that utilize structured sparsity are called structured pruning methods.

2:4 SPARSITY

- Below is an example of 2:4 sparse representation of a matrix.
- We can reduce the number of columns by half,
  - But we also need to keep track of the indices of the non-zero elements within each block of 4 contiguous elements.



[Pool et al., Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT, 2021]

2:4 SPARSITY

- This representation is more amenable to block matrix multiplication methods.
- It's easier to exploit optimizations in the GPU.

Input Operands	Accumulator	Dense TOPS	vs. FFMA	Sparse TOPS	vs. FFMA
FP32	FP32	19.5	–	**–**	**–**
TF32	FP32	156	8X	**312**	**16X**
FP16	FP32	312	16X	**624**	**32X**
BF16	FP32	312	16X	**624**	**32X**
FP16	FP16	312	16X	**624**	**32X**
INT8	INT32	624	32X	**1248**	**64X**

[Pool et al., Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT, 2021]

K:N STRUCTURED PRUNING

- We can adapt magnitude pruning and Wanda to produce $k{:}n$ structured matrices.
- For example, in 2:4 magnitude pruning, we inspect each contiguous block of 4 matrix elements
  - Select the two elements with smallest magnitude and set them to zero.
- In Wanda, we do the same, except we select the two elements with the smallest importance value.
- Sun et al. (2024) measured the accuracy of 2:4 and 4:8 sparsity using magnitude pruning, SparseGPT (Frantar and Alistarh, 2023), and Wanda,
  - On the same 7 zero-shot tasks as before.

K:N STRUCTURED PRUNING

			LLaMA				LLaMA-2		
Method	Weight Update	Sparsity	7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	59.99	62.59	65.38	66.97	59.71	63.03	67.08
Magnitude	✗	50%	46.94	47.61	53.83	62.74	51.14	52.85	60.93
SparseGPT	✓	50%	**54.94**	58.61	63.09	66.30	**56.24**	60.72	**67.28**
Wanda	✗	50%	54.21	**59.33**	**63.60**	**66.67**	**56.24**	**60.83**	67.03
Magnitude	✗	4:8	46.03	50.53	53.53	62.17	50.64	52.81	60.28
SparseGPT	✓	4:8	**52.80**	55.99	60.79	64.87	**53.80**	**59.15**	65.84
Wanda	✗	4:8	52.76	**56.09**	**61.00**	**64.97**	52.49	58.75	**66.06**

[Sun et al., 2024]

K:N STRUCTURED PRUNING

			LLaMA				LLaMA-2		
Method	Weight Update	Sparsity	7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	59.99	62.59	65.38	66.97	59.71	63.03	67.08
Magnitude	✗	50%	46.94	47.61	53.83	62.74	51.14	52.85	60.93
SparseGPT	✓	50%	**54.94**	58.61	63.09	66.30	**56.24**	60.72	**67.28**
Wanda	✗	50%	54.21	**59.33**	**63.60**	**66.67**	**56.24**	**60.83**	67.03
Magnitude	✗	4:8	46.03	50.53	53.53	62.17	50.64	52.81	60.28
SparseGPT	✓	4:8	**52.80**	55.99	60.79	64.87	**53.80**	**59.15**	65.84
Wanda	✗	4:8	52.76	**56.09**	**61.00**	**64.97**	52.49	58.75	**66.06**
Magnitude	✗	2:4	44.73	48.00	53.16	61.28	45.58	49.89	59.95
SparseGPT	✓	2:4	**50.60**	**53.22**	58.91	62.57	**50.94**	54.86	63.89
Wanda	✗	2:4	48.53	52.30	**59.21**	**62.84**	48.75	**55.03**	**64.14**

[Sun et al., 2024]

K:N STRUCTURED PRUNING

- Sun et al. (2024) measured the speed of transformer operations in Llama-65B.
  - They compared operations on dense matrices vs 2:4 sparse matrices.
  - Computation times are shown in milliseconds (on A6000 GPUs).
  - `q/k/v/o_proj` are the linear layers in the attention matrix (the projections that produce the query, key, value, and output matrices).
  - `up/gate_proj` and `down_proj` are the two linear layers in the FF block.

LLaMA Layer	Dense	2:4	Speedup
`q/k/v/o_proj`	3.49	2.14	1.63×
`up/gate_proj`	9.82	6.10	1.61×
`down_proj`	9.92	6.45	1.54×

APPROXIMATION COST OF PRUNING

- While 2:4 pruning effectively halves the memory footprint of the model,
  - And improves its speed by ~1.6x,
  - There is an approximation cost, in terms of accuracy and perplexity.
- One idea to mitigate this approximation cost:
  - After pruning, fine-tune the model.
- Sun et al. (2024) ran this experiment where Llama-7B was fine-tuned on the C4 dataset (unstructured text data for language model training).

APPROXIMATION COST OF PRUNING

- With full fine-tuning, they were able to reduce the approximation cost considerably (though there is still a gap).
- But full fine-tuning is expensive, and not practical for large models without significant hardware resources.

Evaluation	Dense	Fine-tuning	50%	4:8	2:4
Zero-Shot	59.99	✗	54.21	52.76	48.53
		Full	**58.15**	**56.65**	**56.19**

[Sun et al., 2024]

APPROXIMATION COST OF PRUNING

- Parameter-efficient fine-tuning methods, such as LoRA, can be used instead.
- While not as effective as fine-tuning at reducing the approximation gap,
- It is much more feasible to do with limited hardware resources.

Evaluation	Dense	Fine-tuning	50%	4:8	2:4
Zero-Shot	59.99	✗	54.21	52.76	48.53
		LoRA	56.53	54.87	54.46
		Full	**58.15**	**56.65**	**56.19**
Perplexity	5.68	✗	7.26	8.57	11.53
		LoRA	6.84	7.29	8.24
		Full	**5.98**	**6.63**	**7.02**

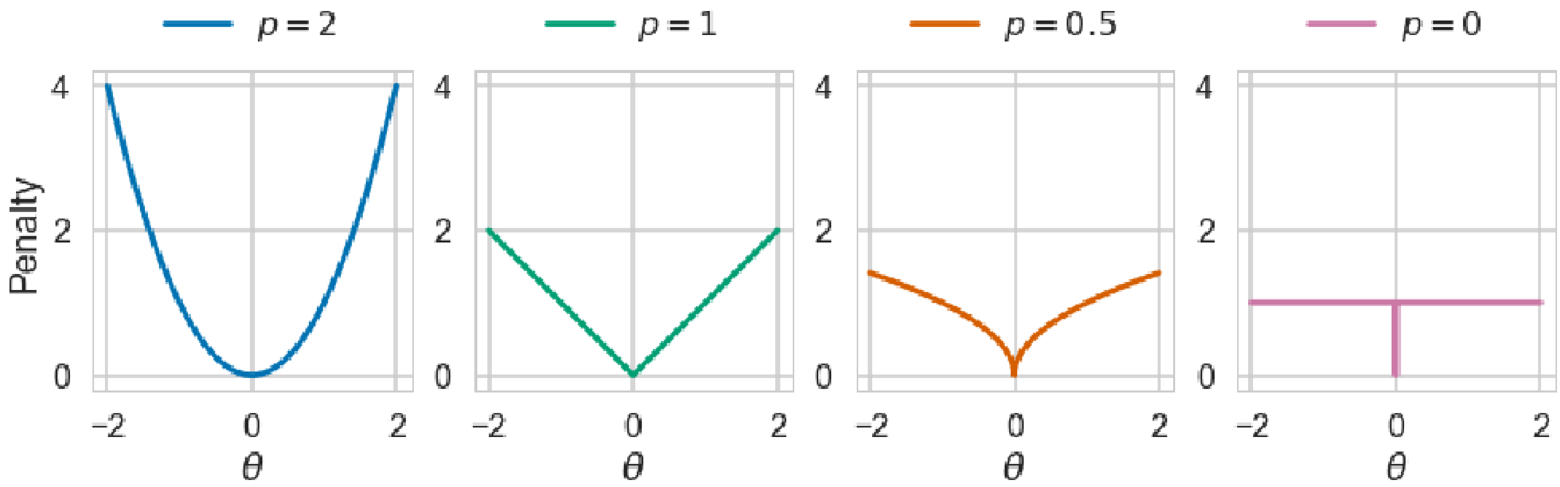
[Sun et al., 2024]

LEARNING WHICH PARAMETERS TO PRUNE

- Suppose I have a model with n dense parameters.
- I want to prune it so that I only have k non-zero parameters.
- We have previously discussed methods that inspect the weights/activations after pretraining and decide which k parameters are most "important."
  - These methods produce an approximation cost.
- Can we have the training/fine-tuning select which parameters to keep/prune?
- One idea: L0 regularization.

L0 REGULARIZATION

- L0 regularization is a special case of Lp regularization.
  - L0 regularization is the limit of Lp regularization as p goes to 0.
  - $||x||_0 = 0$ if $x = 0$, otherwise $||x||_0 = 1$.
- This is useful for counting the number of non-zero parameters in a model.



[Louizos et al., 2018]

L0 REGULARIZATION

- L0 regularization is a special case of Lp regularization.
  - L0 regularization is the limit of Lp regularization as p goes to 0.
  - $||x||_0 = 0$ if $\mathtt{x} = 0$, otherwise $||x||_0 = 1$.
- This is useful for counting the number of non-zero parameters in a model.
- If we have model with parameters θ and loss function $L(\theta)$,
  - We can add a constraint to the loss function:

$$\arg\min_{\theta} L(\theta) \text{ subject to } \sum_i ||\theta_i||_0 = k.$$

- Using Lagrange multipliers, we can rewrite this as:

$$\arg\min_{\theta,\lambda} L(\theta) + \lambda(k - \sum_i ||\theta_i||_0).$$

- Problem: The objective function is not differentiable.

L0 REGULARIZATION

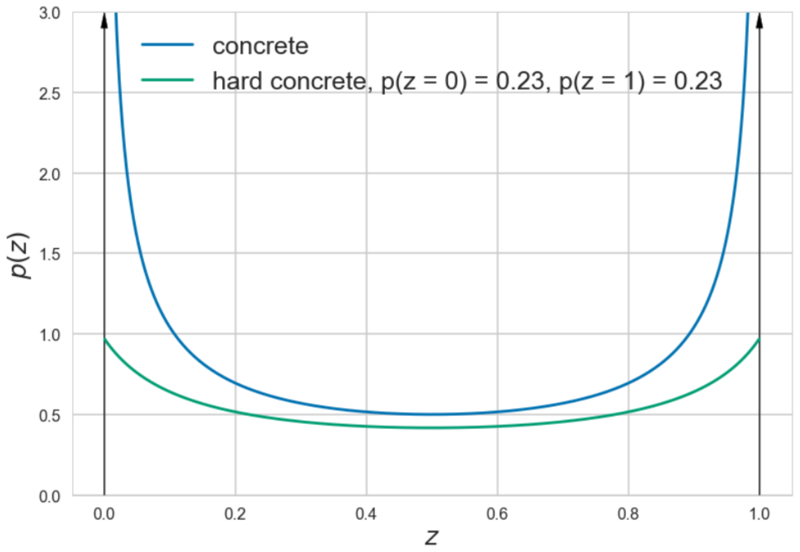
- Solution: Introduce a mask variable z_i for each parameter θ_i:
  - Each z_i is either 0 or 1.
  - Each parameter θ_i is multiplied by z_i in the model.
  - Effectively, we are pruning the parameters θ_i where $z_i = 0$.
- So our objective function is now:

$$\arg\min_{\theta,z} \max_{\lambda} L(\theta \odot z) + \lambda(k - \textstyle\sum_i z_i)$$

- But: z is discrete.
  - This is a combinatorial optimization problem.

L0 REGULARIZATION

- Instead, we relax the variable z_i to take any continuous value in $[0,1]$.
- $z_i \sim \mathrm{HardConcrete}(\alpha_i, \beta_i)$
- "Re-parameterization" trick.
- This is a distribution on $[0,1]$ such that $p(z_i = 0) > 0$ and $p(z_i = 1) > 0$.



[Louizos et al., 2018]

L0 REGULARIZATION

- Instead, we relax the variable z_i to take any continuous value in $[0,1]$.
- $z_i \sim \mathrm{HardConcrete}(\alpha_i, \beta_i)$
- "Re-parameterization" trick.
- This is a distribution on $[0,1]$ such that $p(z_i = 0) > 0$ and $p(z_i = 1) > 0$.
- The hard concrete distribution is defined via the following sampling procedure:
  - $u_i \sim \mathrm{Uniform}(0,1)$
  - $s_i' = \sigma((\log u_i - \log(1 - u_i) + \alpha_i)/\beta)$
  - $s_i = s_i'(1.2) - 0.1$
  - $z_i = \min(1, \max(0, s_i))$
- If α_i is large, z_i will tend to be close to 1. If α_i is small, z_i will be close to 0.

L0 REGULARIZATION

- So the loss function is now:

$$\arg\min_{\theta} \max_{\lambda} \mathbb{E}_{z}[L(\theta \odot z) + \lambda(k - \sum_i z_i)]$$

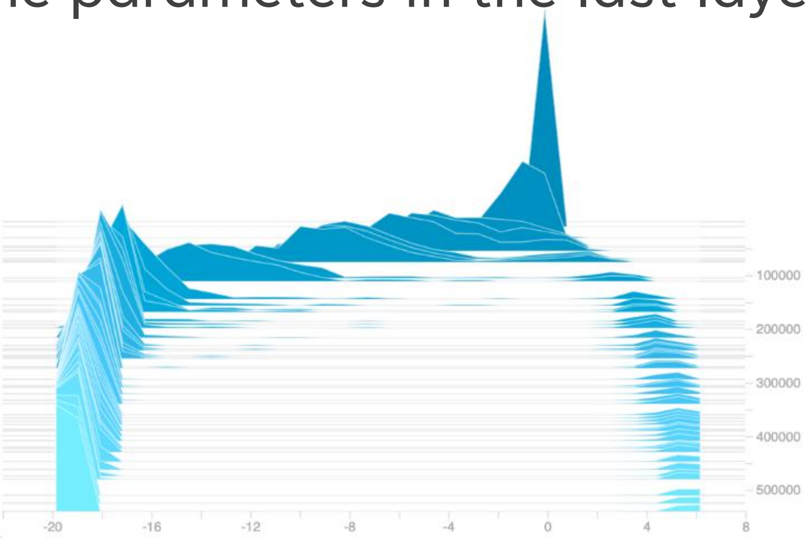
$$= \arg\min_{\theta} \max_{\lambda} \mathbb{E}_{u}[L(\theta \odot z)] + \lambda(k - \sum_i \mathbb{E}_u[z_i])$$

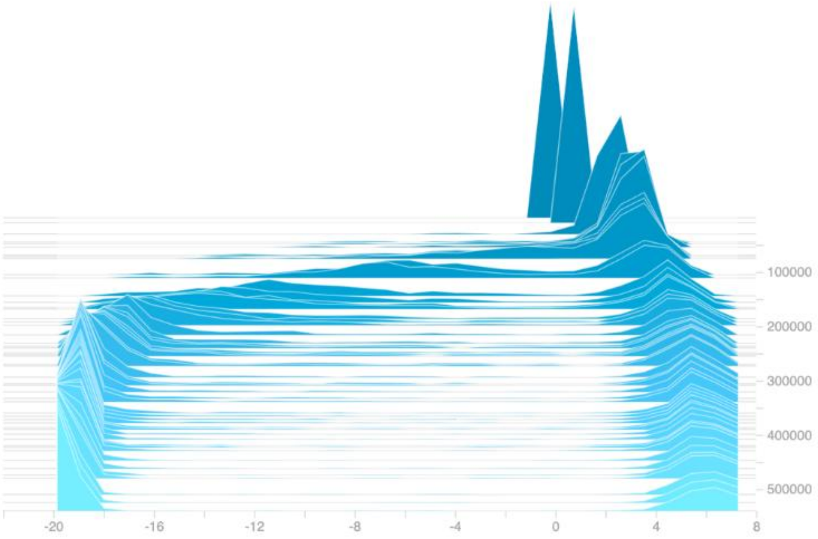
$$= \arg\min_{\theta,\alpha} \max_{\lambda} \mathbb{E}_{u}[L(\theta \odot z)] + \lambda \sum_i \sigma(\alpha_i - \beta \log(0.1/1.1))$$

- We approximate the first term $\mathbb{E}_{u}[L(\theta \odot z)]$ using Monte Carlo sampling:
  - Randomly generate N samples of u,
  - For each sample, compute the loss function $L(\theta \odot z)$,
  - Compute the average of the losses.
- This loss function is differentiable, and we can use gradient descent to simultaneously train the model and learn the mask variables z_i.

L0 REGULARIZATION

- A real-world example of this technique applied to pruning the parameters of a language model.
- We see that as training progresses, the mask variables tend to 0 or 1:
    - The left plot is for the parameters in the first layer and the right plot is for the parameters in the last layer.

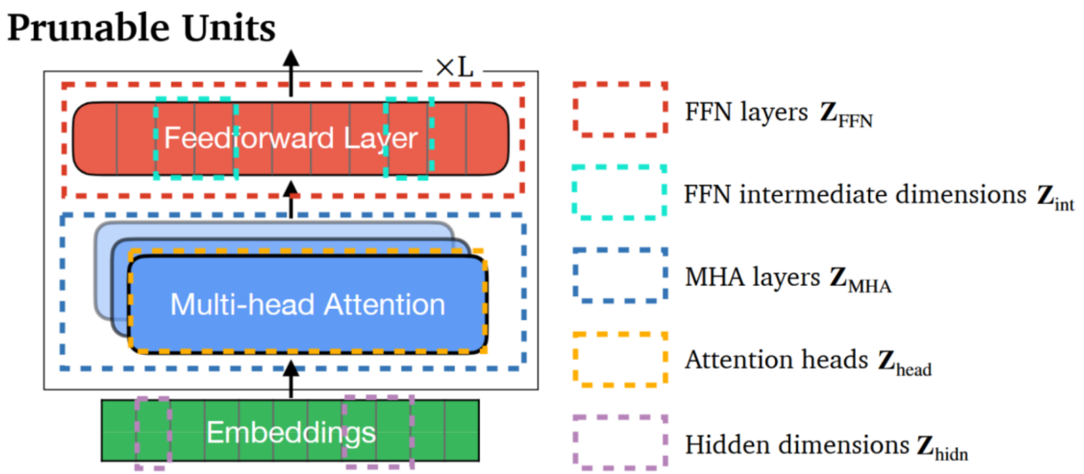




[Wang and Wohlwend and Lei, 2021]

L0 REGULARIZATION OF TRANSFORMER COMPONENTS

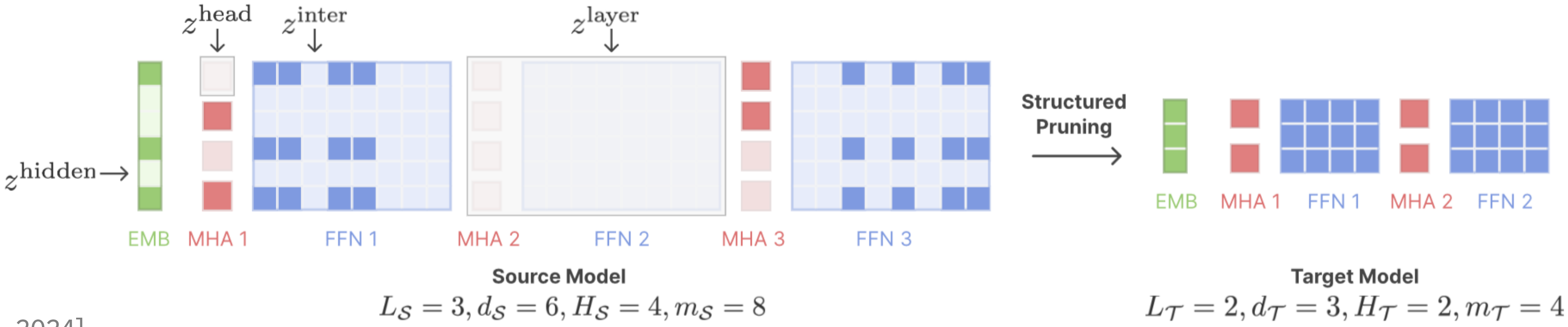
- We can apply this technique on transformer components.
- For each dimension of the embedding, we can introduce a mask variable.
- For each dimension in the FF layer, we can introduce another mask variable.
- We can introduce a mask variable for each attention head in every layer.
- As well as a mask variable for each layer.



[Xia et al., 2024]

L0 REGULARIZATION OF TRANSFORMER COMPONENTS

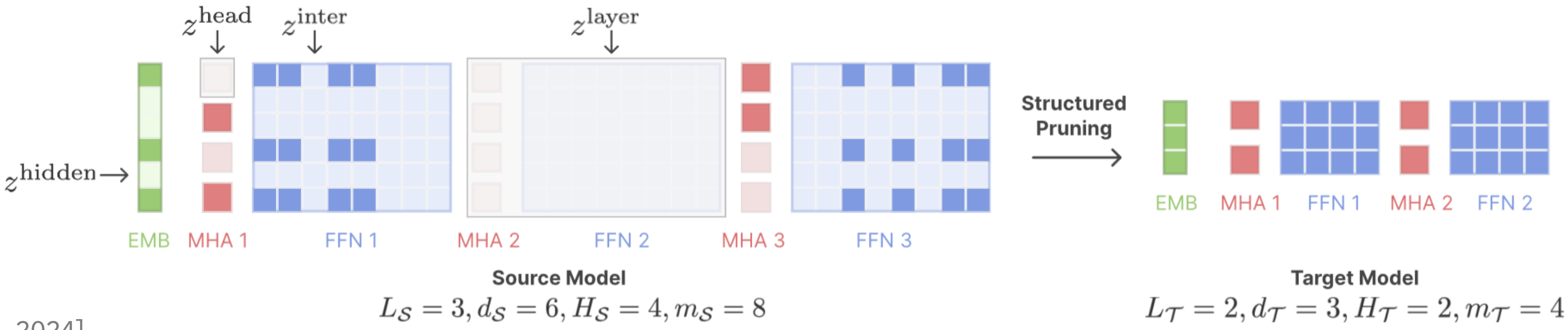
- For this approach, we need a target number of non-zero mask variables.
  - So we need a target number of non-zero embedding dimensions,
  - A target number of non-zero FF dimensions,
  - A target number of attention heads and layers.
- These values can be obtained from existing smaller pretrained models.



[Xia et al., 2024]

L0 REGULARIZATION OF TRANSFORMER COMPONENTS

- Xia et al. (2024) started with Llama2-7B as the source model and used the architecture of Pythia-1.4B as the target architecture.
  - (as a reference for the target embedding dimension, FF dimension, number of attention heads and layers)
- Then they use the reparameterization trick on the mask variables to train the model and learn which transformer components to prune.



[Xia et al., 2024]

L0 REGULARIZATION OF TRANSFORMER COMPONENTS

- Xia et al. (2024) started with Llama2-7B as the source model and used the architecture of Pythia-1.4B as the target architecture.
  - (as a reference for the target embedding dimension, FF dimension, number of attention heads and layers)
- Then they use the reparameterization trick on the mask variables to train the model and learn which transformer components to prune.
  - This is another example of structured pruning.
- The result is a 1.4B parameter model that was obtained from Llama2-7B.
- They repeated their method using INCITE-Base-3B as the target architecture to obtain a 3B parameter model.

SHEARED-LLAMA

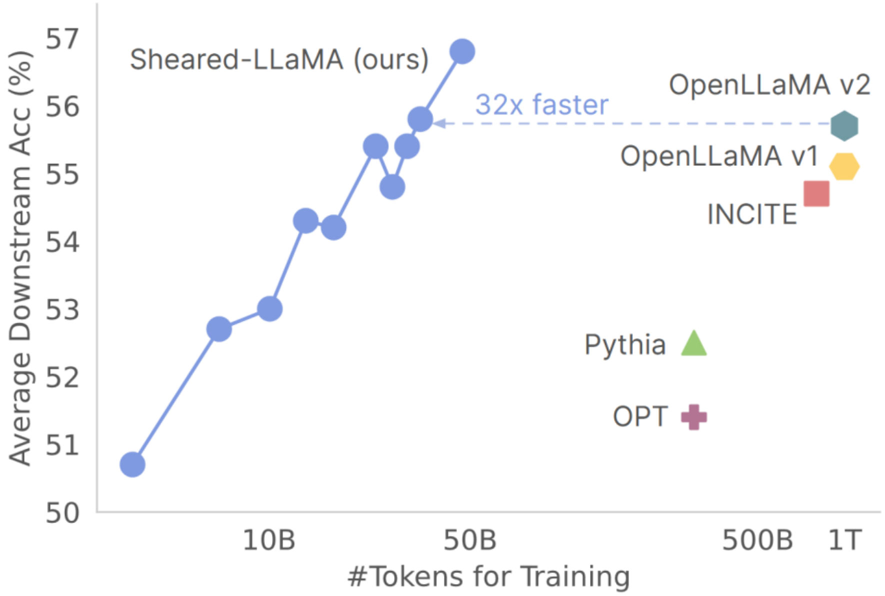
- They called the resulting models Sheared-Llama.
- In order to prune Llama, they trained on a dataset with 50B tokens.
- This is a lot, but smaller compared to pretraining a model from scratch:

Model	Pre-training Data	#Tokens
LLaMA1	LLaMA data	1T
LLaMA2	*Unknown*	2T
OPT	OPT data[5]	300B
Pythia	The Pile	300B
INCITE-Base	RedPajama	800B
OpenLLaMA v1	RedPajama	1T
OpenLLaMA v2	OpenLLaMA data[6]	1T
TinyLlama	TinyLlama data[7]	3T
Sheared-LLaMA	RedPajama	50B

[Xia et al., 2024]

SHEARED-LLAMA

- They also found that the resulting models were more accurate than either Pythia or INCITE.
- They ran each model on 11 downstream benchmarks and compared their average accuracies.



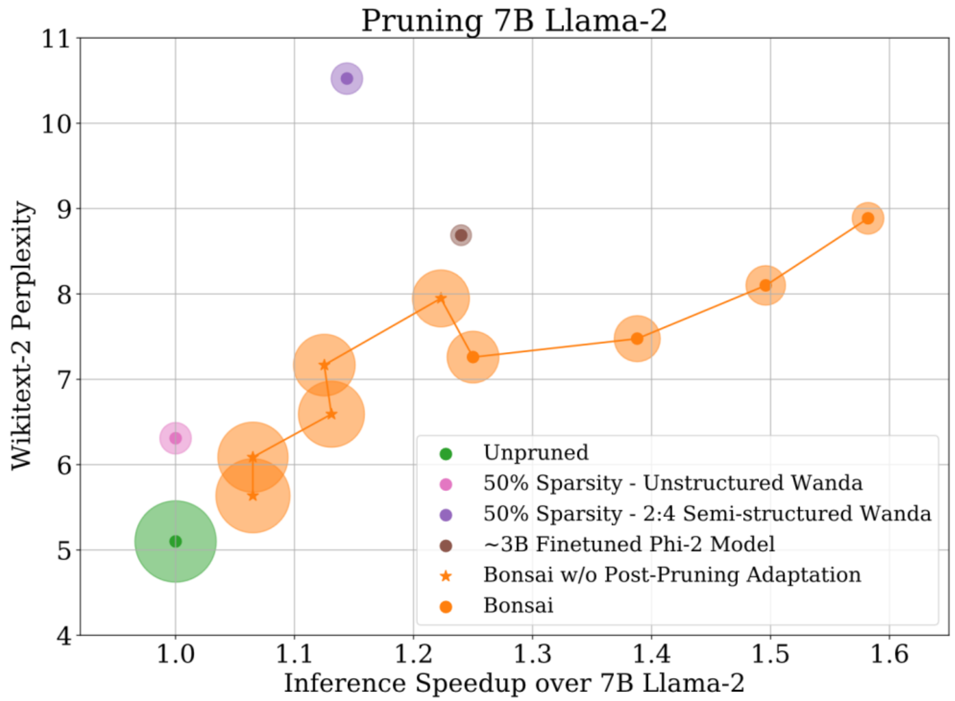
[Xia et al., 2024]

PRUNING VIA L0 REGULARIZATION

- Pruning via L0 regularization and training can be expensive.
  - Especially in terms of memory (we need to store gradients for the mask variables).
- Can we avoid training?
- What if the model is so large that we can only do forward passes?
- One idea is to use forward passes to estimate the “relevance” of various model components.
  - As before, the model components can be attention heads, FF dimensions, embeddings dimensions, or even entire layers.
- Once we have an estimated relevance value for each component, prune the components with the lowest relevance.

PRUNING VIA FORWARD PASSES

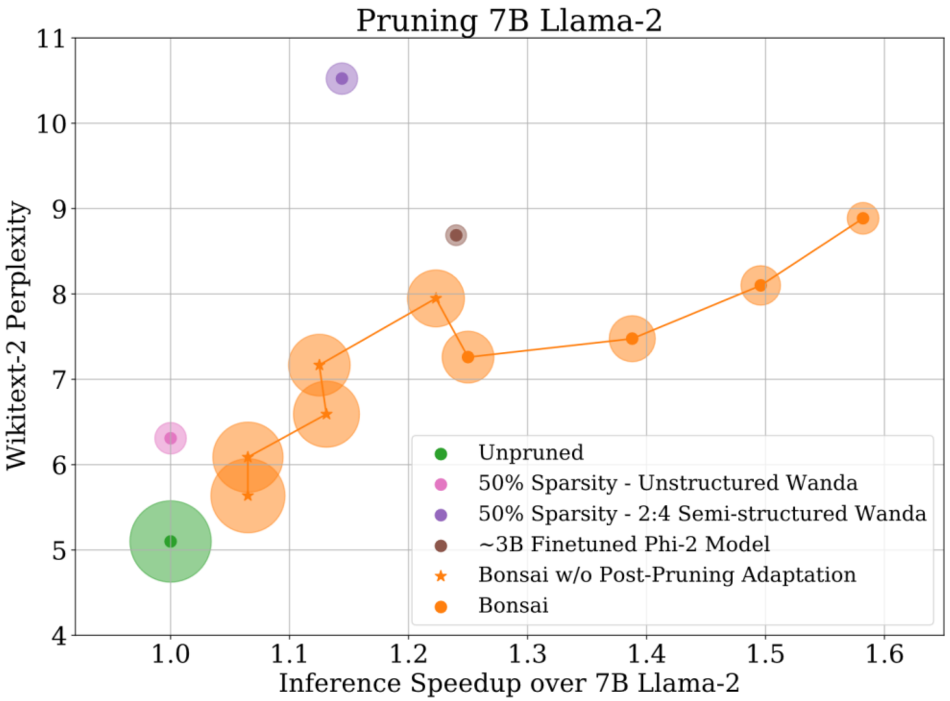
- This approach was proposed by Dery et al. (2024) and is called Bonsai.
- The size of the circle indicates the model size.



[Dery et al., 2024]

PRUNING VIA FORWARD PASSES

- The resulting pruned model can be made significantly faster than 2:4 structured Wanda,
  - But is also more accurate (lower perplexity).



[Dery et al., 2024]

EFFECTS OF PRUNING

- We find that pruning generally causes decrease in model performance on downstream tasks.
- But what about other aspects of the model?
  - Out-of-distribution performance?
  - Hallucination frequency?
  - etc...
- Much remains unexplored.
- Chrysostomou and Zhao and Williams et al. (2024) examined some of these other properties of pruned models (using SparseGPT and Wanda).

EFFECTS OF PRUNING

- Chrysostomou and Zhao and Williams et al. (2024) used automated metrics to quantify hallucination risk.
- They focus on the summarization task:
  - Given a document, output a short summary that contains all the important high-level information in the given document.
- Then they compute the hallucination risk ratio,
  - Where a ratio of less than 1 indicates the pruned model has lower hallucination risk than the original model.
  - A ratio of greater than 1 indicates the pruned model has greater hallucination risk than the original model.

EFFECTS OF PRUNING

		Llama-2 7B				Llama-2 13B				Llama-2 70B				Mistral 7B				OPT-IML 30B			
		SparseGPT		Wanda		SparseGPT		Wanda		SparseGPT		Wanda		SparseGPT		Wanda		SparseGPT		Wanda	
Dataset	Metric	2:4	50%	2:4	50%	2:4	50%	2:4	50%	2:4	50%	2:4	50%	2:4	50%	2:4	50%	2:4	50%	2:4	50%
FactCC	HaRiM^{+}	**0.98**	**0.95**	**0.94**	**0.95**	**0.77**	**0.95**	**0.69**	**0.91**	**0.93**	**0.96**	**0.93**	**0.96**	**0.93**	**0.94**	**0.91**	**0.94**	**0.83**	**0.87**	**0.87**	**0.85**
	SummaC$_{conv}$	**0.64**	**0.82**	**0.56**	**0.81**	**0.76**	**0.83**	**0.64**	**0.84**	**0.76**	**0.92**	**0.77**	**0.90**	**0.79**	**0.88**	**0.74**	**0.86**	**0.80**	**0.86**	**0.84**	**0.83**
	SummaC$_{zs}$	**0.47**	**0.65**	**0.39**	**0.65**	**0.50**	**0.61**	**0.41**	**0.61**	**0.63**	**0.86**	**0.63**	**0.83**	**0.76**	**0.85**	**0.68**	**0.82**	**0.80**	**0.87**	**0.85**	**0.83**
Polytope	HaRiM^{+}	**0.97**	**0.97**	**0.97**	**0.97**	**0.78**	**0.93**	**0.71**	**0.85**	**0.94**	0.96	**0.95**	1.00	**0.95**	**0.95**	**0.94**	**0.96**	**0.87**	**0.93**	**0.92**	**0.88**
	SummaC$_{conv}$	**0.67**	**0.83**	**0.69**	**0.83**	**0.70**	**0.78**	**0.65**	**0.79**	**0.77**	**0.93**	**0.78**	**0.92**	**0.78**	**0.82**	**0.76**	**0.84**	**0.86**	**0.95**	**0.91**	**0.92**
	SummaC$_{zs}$	**0.64**	**0.85**	**0.64**	**0.75**	**0.58**	**0.69**	**0.56**	**0.69**	**0.75**	**0.88**	**0.74**	**0.83**	**0.76**	**0.81**	**0.75**	**0.84**	**0.88**	**0.95**	**0.92**	**0.93**
SummEval	HaRiM^{+}	**0.88**	**0.93**	**0.81**	**0.93**	**0.80**	**0.97**	**0.69**	**0.96**	**0.95**	0.98	**0.95**	0.98	**0.93**	**0.94**	**0.92**	**0.95**	**0.91**	**0.92**	**0.90**	**0.89**
	SummaC$_{conv}$	**0.55**	**0.81**	**0.46**	**0.76**	**0.67**	**0.81**	**0.59**	**0.81**	**0.78**	0.96	**0.79**	0.93	**0.79**	**0.85**	**0.77**	**0.87**	**0.86**	**0.88**	**0.83**	**0.85**
	SummaC$_{zs}$	**0.49**	**0.75**	**0.4**	**0.68**	**0.56**	**0.71**	**0.49**	**0.66**	**0.70**	**0.92**	**0.70**	**0.88**	**0.79**	**0.84**	**0.76**	**0.88**	**0.86**	**0.89**	**0.85**	**0.86**
Legal Contracts	HaRiM^{+}	**0.99**	**0.85**	**0.90**	**0.85**	**0.83**	**0.88**	**0.76**	**0.88**	**0.87**	**0.92**	**0.89**	**0.95**	**0.85**	**0.94**	**0.89**	**0.93**	**0.85**	**0.89**	**0.81**	**0.83**
	SummaC$_{conv}$	0.98	0.85	0.93	0.94	**0.82**	**0.81**	**0.76**	**0.81**	**0.79**	**0.88**	**0.83**	**0.91**	**0.83**	**0.92**	**0.92**	**0.89**	**0.85**	**0.88**	**0.81**	**0.86**
	SummaC$_{zs}$	1.01	**0.86**	0.96	**0.90**	**0.93**	**0.86**	**0.88**	**0.88**	**0.85**	0.93	**0.88**	0.95	**0.88**	**0.92**	**0.93**	**0.92**	**0.93**	0.96	**0.94**	1.00
RCT	HaRiM^{+}	**0.92**	**0.96**	**0.87**	**0.92**	**0.86**	**0.99**	**0.80**	**0.97**	**0.93**	**0.96**	**0.93**	**0.97**	**0.93**	**0.96**	**0.93**	**0.95**	**0.85**	**0.88**	**0.83**	**0.87**
	SummaC$_{conv}$	**0.69**	**0.86**	**0.70**	**0.88**	**0.78**	**0.89**	**0.79**	**0.88**	**0.82**	**0.92**	**0.82**	**0.93**	**0.82**	**0.88**	**0.81**	**0.87**	**0.83**	**0.88**	**0.79**	**0.88**
	SummaC$_{zs}$	**0.71**	**0.83**	**0.71**	**0.82**	**0.69**	**0.81**	**0.70**	**0.82**	**0.79**	**0.90**	**0.79**	**0.90**	**0.84**	**0.89**	**0.82**	**0.89**	**0.77**	**0.80**	**0.77**	**0.83**
Average	HaRiM^{+}	0.95	0.93	0.90	0.92	0.81	0.95	0.73	0.91	0.92	0.96	0.93	0.97	0.92	0.95	0.92	0.95	0.87	0.90	0.87	0.87
	SummaC$_{conv}$	0.70	0.83	0.67	0.85	0.74	0.82	0.68	0.83	0.78	0.92	0.80	0.92	0.80	0.87	0.80	0.87	0.84	0.89	0.84	0.87
	SummaC$_{zs}$	0.67	0.79	0.62	0.76	0.65	0.74	0.61	0.73	0.74	0.90	0.75	0.88	0.81	0.86	0.79	0.87	0.85	0.90	0.86	0.89

[Chrysostomou and Zhao and Williams et al., 2024]

EFFECTS OF PRUNING

- Interestingly, pruned models are less likely to hallucinate.
- 2:4 structured pruned models are less likely to hallucinate than 50% unstructured pruned models.
- The reduction in hallucination risk was larger in the smaller Llama models, as compared to Llama2-70B, Mistral-7B, and OPT-IML-30B.

EFFECTS OF PRUNING

- They also performed human evaluation on these models to validate the results from their automated hallucination risk metrics.
  - They gave each human evaluator 100 news articles as well as summaries from both the pruned model and original model.
  - Each evaluator was tasked to annotate:
    - Which summary has more hallucinations?
    - Which summary has more omissions?
    - Which summary has more repetitive information?
    - Which summary is more semantically aligned with the original article?

EFFECTS OF PRUNING

- Inter-annotator agreement (IAA) is measured using Cohen's kappa score.
  - A score closer to 1 indicates better agreement.
- Humans also find that the pruned model produces fewer hallucinations.

Model	Halluc. Q1 (↓)	Omiss. Q2 (↓)	Repet. Q3 (↓)	Align. Q4 (↑)
Llama-2 7B	31	**5**	**0**	**28**
w/ SparseGPT	**14**	18	9	21
IAA (κ)	0.82	0.63	0.62	0.53
Mistral 7B	12	**9**	**0**	**31**
w/ SparseGPT	**10**	13	5	23
IAA (κ)	0.87	0.61	0.67	0.59

[Chrysostomou and Zhao and Williams et al., 2024]

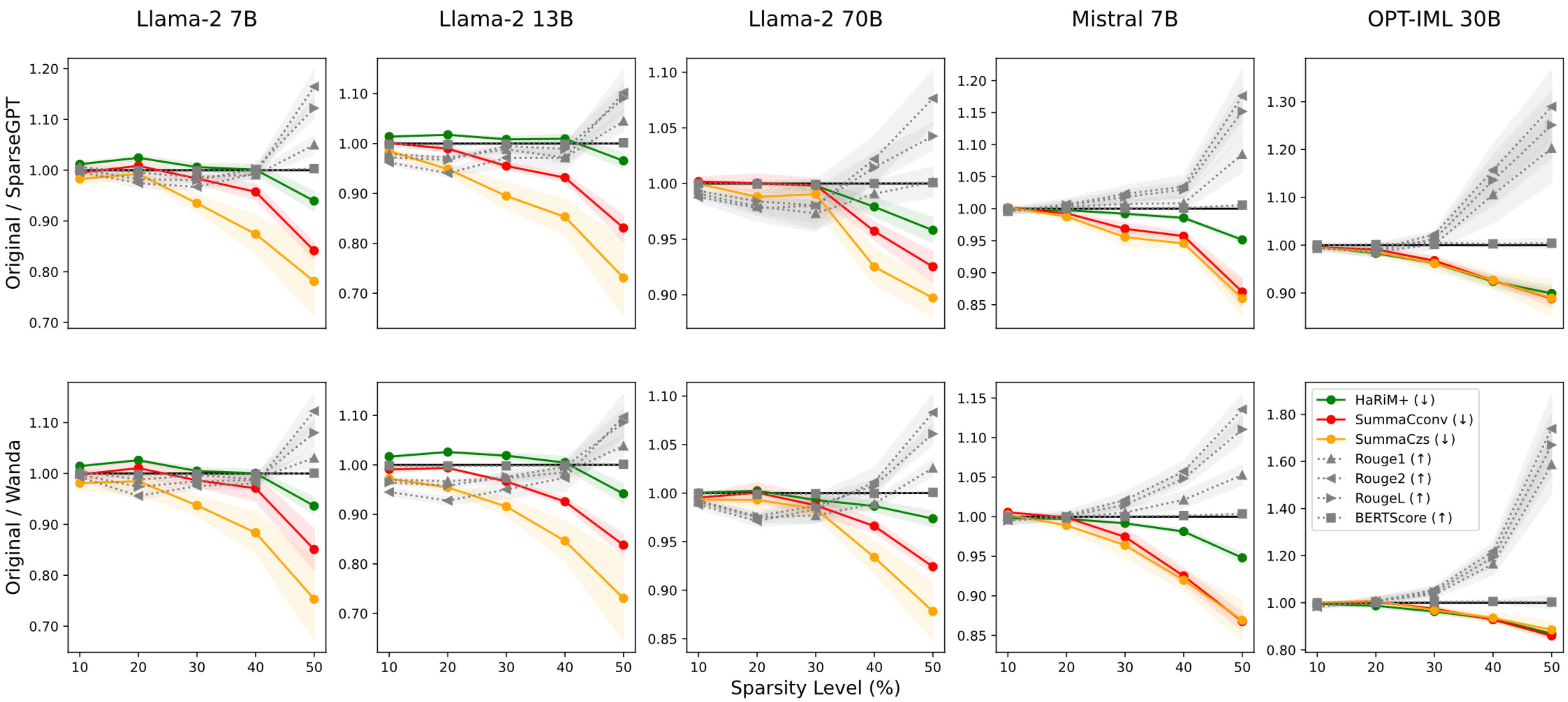
EFFECTS OF PRUNING

- But humans find that the pruned models omitted important information more often.
  - And the pruned models had more repetition.

Model	Halluc. Q1 (↓)	Omiss. Q2 (↓)	Repet. Q3 (↓)	Align. Q4 (↑)
Llama-2 7B	31	**5**	**0**	**28**
w/ SparseGPT	**14**	18	9	21
IAA (κ)	0.82	0.63	0.62	0.53
Mistral 7B	12	**9**	**0**	**31**
w/ SparseGPT	**10**	13	5	23
IAA (κ)	0.87	0.61	0.67	0.59

[Chrysostomou and Zhao and Williams et al., 2024]

HALLUCINATION RISK VS SPARSITY



[Chrysostomou and Zhao and Williams et al., 2024]

PRUNING SUMMARY

- In this lecture, we discussed pruning as a strategy to reduce the memory footprint of large models.
  - Pruning can also be used to make models faster.
  - But this is less likely for unstructured pruning methods.
- Structured pruning can produce models that are both smaller and faster.
- But there is an approximation cost.
- Next time, we will finish the lecture series on model compression and discuss model distillation.

QUESTIONS?